The RIGHT model for Continuous Experimentation

Fabian Fagerholm[a,*], Alejandro Sanchez Guinea[b], Hanna Mäenpää[a], Jürgen Münch[a,c]

[a] *Department of Computer Science, University of Helsinki, P.O. Box 68, FI-00014 University of Helsinki, Finland*

[b] *University of Luxembourg, 4 rue Alphonse Weicker, L-2721, Luxembourg*

[c] *Faculty of Informatics, Reutlingen University, Alteburgstraße 150, D-72762 Reutlingen, Germany*

Abstract

Context: Development of software-intensive products and services increasingly occurs by continuously deploying product or service increments, such as new features and enhancements, to customers. Product and service developers must continuously find out what customers want by direct customer feedback and usage behaviour observation. **Objective:** This paper examines the preconditions for setting up an experimentation system for continuous customer experiments. It describes the RIGHT model for Continuous Experimentation (Rapid Iterative value creation Gained through High-frequency Testing), illustrating the building blocks required for such a system. **Method:** An initial model for continuous experimentation is analytically derived from prior work. The model is matched against empirical case study findings from two startup companies and further developed. **Results:** Building blocks for a continuous experimentation system and infrastructure are presented. **Conclusions:** A suitable experimentation system requires at least the ability to release minimum viable products or features with suitable instrumentation, design and manage experiment plans, link experiment results with a product roadmap, and manage a flexible business strategy. The main challenges are proper, rapid design of experiments, advanced instrumentation of software to collect, analyse, and store relevant data, and the integration of experiment results in both the product development cycle and the software development process.

*Corresponding author

Email addresses: fabian.fagerholm@helsinki.fi (Fabian Fagerholm), alejandro.sanchezguinea@uni.lu (Alejandro Sanchez Guinea), hanna.maenpaa@cs.helsinki.fi (Hanna Mäenpää), juergen.muench@cs.helsinki.fi, juergen.muench@reutlingen-university.de (Jürgen Münch)

Keywords: Continuous experimentation, Product development, Software architecture, Software development process, Agile software development, Lean software development,

1. Introduction

The accelerating digitalisation in most industry sectors means that an increasing number of companies are or will soon be providers of software-intensive products and services. Simultaneously, new companies already enter the marketplace as software companies. Software enables increased flexibility in the types of services that can be delivered, even after an initial product has been delivered to customers. Many constraints that previously existed, particularly in terms of the behaviour of a product or service, can now be removed.

With this newfound flexibility, the challenge for companies is no longer primarily how to identify and solve technical problems, but rather how to solve problems which are relevant for customers and thereby deliver value. Finding solutions to this problem has often been haphazard and based on guesswork, but many successful companies have approached this issue in a systematic way. Recently, a family of generic approaches has been proposed. For example, the Lean Startup methodology [26] proposes a three-step cycle: build, measure, learn.

However, a detailed framework for conducting systematic, experiment-based software development has not been elaborated. Such a framework has implications for the technical product infrastructure, the software development process, the requirements regarding skills that software developers need to design, execute, analyse, and interpret experiments, and the organisational capabilities needed to operate and manage a company based on experimentation in research and development.

Methods and approaches for continuous experimentation with software product and service value should itself be based on empirical research. In this paper, we present the most important building blocks of a framework for continuous experimentation. Specifically, our research question is:

RQ How can Continuous Experimentation with software-intensive products and services be organised in a systematic way?

To further scope the question, we split it into two sub-questions:

RQ1 What is a suitable process model for Continuous Experimentation with software-intensive products and services?

RQ2 What is a suitable infrastructure architecture for Continuous Experimentation with software-intensive products and services?

We give an answer to the research questions by validating an analytically derived model against a series of case studies in which we implemented different parts of the model in cooperation with two startup companies. The result is the RIGHT model for Continuous Experimentation (Rapid Iterative value creation Gained through High-frequency Testing). This model focuses on developing the *right* software, whereas the typical focus of software engineering in the past has been on developing the software right (e.g. in terms of technical quality). The model is instantiated in the RIGHT process model and the RIGHT infrastructure architecture model. Together, these instantiations address the need to integrate the requirements, design, implementation, testing, deployment, and maintenance phases of software development in a way that uses continuous empirical feedback from users.

The rest of this paper is organised as follows. In Section 2, we review related work on integrating experimentation into the software development process. In Section 3, we describe the research approach and context of the study. In Section 4, we first present our proposed model for continuous experimentation, and then relate the findings of our case study to it in order to illustrate its possible application and show the empirical observations that it was grounded in. In Section 5, we discuss the model and consider some possible variations. Finally, we conclude the paper and present an outlook on future work in Section 6.

2. Related work

Delivering software that has value – utility for its users – can be considered a primary objective for software development projects. In this section, we describe models for systematic value delivery and approaches for using experiments as a means for value testing and creation. In addition, we discuss related work with respect to experiments at scale.

2.1. Models for systematic value delivery

Lean manufacturing and the Toyota Production System [22] has inspired the definition of Lean software development. This approach provides comprehensive guidance for the combination of design, development, and validation built as a single feedback loop focused on discovery and delivery of value [25]. The main ideas of this approach, which have been emphasised since its introduction, are summarised in seven principles: optimize the whole, eliminate waste, build quality in, learn constantly, deliver fast, engage everyone, and keep getting better [24].

Lean Startup [26] provides mechanisms to ensure that product or service development effectively addresses what customers want. The methodology is based on the Build-Measure-Learn loop that establishes learning about customers and their needs as the unit of progress. It proposes to apply scientific method and thinking to startup businesses in the form of learning experiments. As the results of experiments are analysed, the company has to decide to "persevere" on the same path or "pivot" in a different direction while considering what has been learned from customers.

Customer Development [4] emphasises the importance of not only doing product development activities but also to learn and discover who a company's initial customers will be, and what markets they are in. Customer Development argues that a separate and distinct process is needed for those activities. Customer Development is a four-step model divided into a search and an execution phase. In the search phase, a company performs *customer discovery*, testing whether the business model is correct (product/market fit), and *customer validation*, which develops a replicable sales model. In the execution phase, *customer creation* focuses on creating and driving demand, and

company building is the transition from an organisation designed to learn and discover to one that is optimised for cost-efficient delivery of validated products or services.

In light of the benefits that a methodology such as Lean Startup can provide, where controlled experiments constitute the main activity driving development, Holmström Olsson et al. [12] propose a target stage for any company that wishes to build a development system with the ability to continuously learn from real-time customer usage of software. They describe the stages that a company has to traverse in order to achieve that target as the "stairway to heaven". The target stage is achieved when the software organisation functions as an R&D experiment system. The stages on the way to achieving the target are: (i) traditional development, (ii) agile R&D organisation, (iii) continuous integration, and (iv) continuous deployment. The authors first describe these four stages and then analyse them through a multiple-case study that examines the barriers that exist on each step on the path towards continuous deployment. The target stage is only described; the authors do not detail any means to overcome the barriers. A main finding from the case study is that the transition towards Agile development requires shifting to small development teams and focusing on features rather than on components. Also, it is relevant to notice that the transition towards continuous integration requires an automated build and test system (continuous integration), a main version control branch to which code is continuously delivered, and modularised development. Holmström Olsson et al. found that in order to move from continuous integration to continuous deployment, organisational units such as product management must be fully involved, and close work with a very active lead customer is needed when exploring the product concept further. The authors suggest two key actions to make the transition from continuous deployment to an R&D experiment system. First, the product must be instrumented so that field data can be collected in actual use. Second, organisational capabilities must be developed in order to effectively use the collected data for testing new ideas with customers.

Other works have studied some of the stages of the "stairway to heaven" individually. Ståhl & Bosch [27] have studied the continuous integration stage, pointing out that there is no homogeneous practice of continuous integration in the industry. They propose a descriptive model that allows studying and evaluating the different ways in which

continuous integration can be viewed. Eklund & Bosch [7] present an architecture that supports continuous experimentation in embedded systems. They explore the goals of an experiment system, develop experiment scenarios, and construct an architecture that supports the goals and scenarios. The architecture combines an experiment repository, data storage, and software to be deployed on embedded devices via over-the-air data communication channels. The architecture also considers the special requirements for safety in, e.g., automotive applications. However, the main type of experiment is confined to A/B testing, and the architecture is considered mainly from the perspective of a software development team rather than a larger product development organisation.

Holmström Olsson & Bosch [13] describe the Hypothesis Experiment Data-Driven Development (HYPEX) model. The goal of this model is to shorten the feedback loop to customers. It consists of a loop where potential features are generated into a feature backlog, features are selected and a corresponding expected behaviour is defined. The expected behaviour is used to implement and deploy a minimum viable feature (MVF). Observed and expected behaviour is compared using a gap analysis, and if a sufficiently small gap is identified, the feature is finalised. On the other hand, if a significant gap is found, hypotheses are developed to explain it, and alternative MVFs are developed and deployed, after which the gap analysis is repeated. The feature may also be abandoned if the expected benefit is not achieved.

2.2. Systematic value creation through experimentation

The models outlined above all aim to make experimentation systematic in the software development organisation. One important conceptual concern is the definition of experimentation. Experimentation has been established in software engineering since the 1980s. Basili et al. [3] were among the fist to codify a framework and process for experimentation. Juristo et al. [14] and Wohlin et al. [31] present more recent syntheses regarding experimentation in software engineering. Taken together, these works show that "experimentation" in software engineering can be considered in a broad sense, including both controlled experiments but also more explorative activities which aim at understanding and discovery rather than hypothesis testing. For the purposes of this article, we consider experimentation to be a range of activities that can be placed

within a spectrum including controlled experiments as well as open-ended exploration. However, we emphasise that regardless of the placement within this spectrum, all methods require rigorous study designs and have a defensible and transparent way of reasoning and drawing conclusions from empirical data. They are not the same method being applied more or less carefully. The logic of controlled experiments relies on careful manipulation of variables, observation of effects, and analysis to test for causal relationships. Quasi-controlled experiments relax some of the requirements for randomised treatment. Case studies often include qualitative elements and their logic is different from controlled experiments: they generalise analytically rather than statistically [32]. Qualitative methods may also be used alone, such as through interview- or observation-based studies.

Experimentation may also be considered in terms of goals, and goals may exist on different levels of the product development organisation. On the product level, experimentation may be used to select features from a set of proposed features. On the technical level, experimentation may be used to optimise existing features. However, the model presented in this paper links experimentation on the product and technical level to the product vision and strategy on the business level. Experimentation becomes a systemic activity that drives the entire organisation. This allows for focused testing of business hypotheses and assumptions, which can be turned into faster decision-making and reaction to customer needs. Depending on the specific method used, the results of an experiment may suggest new information which should be incorporated into the decision-making process.

2.3. Considerations for running experiments at a large scale

Previous works have presented case studies that exhibit different aspects concerning continuous experimentation. Steiber [28] report on a study of the continuous experimentation model followed by Google, analysing a success story of this approach. Tang et al. [29] describe an overlapping experiment infrastructure, developed at Google, that allows web queries in a search engine to be part of multiple experiments, thus allowing more experiments to be carried out at a faster rate. Adams [1] present a case study

on the implementation of Adobe's Pipeline, a process that is based on the continuous experimentation approach.

Kohavi et al. [16, 17] note that running experiments at large scale requires addressing multiple challenges in three areas: cultural/organisational, engineering, and trustworthiness. The larger organisation needs to learn the reasons for running controlled experiments and the trade-offs between controlled experiments and other methods of evaluating ideas. Even negative experiments should be run, which degrade user experience in the short term, because of their learning value and long-term benefits. When the technical infrastructure supports hundreds of concurrent experiments, each with millions of users, classical testing and debugging techniques no longer apply because there are millions of live variants of the system in production. Instead of heavy up-front testing, Kohavi et al. report having used alerts and post-deployment fixing. The system has also identified many negative features that were avoided despite being advocated by key stakeholders, saving large amounts of money.

Experimentation also has an important relationship with company culture. Kohavi et al. [15] describe a platform for experimentation built and used at Microsoft, noting the cultural challenges involved in using experiment results, rather than opinions from persons in senior positions, as the basis of decisions. They suggest, for example, that one should avoid trying to build features through extensive planning without early testing of ideas, that experiments should be carried out often, that a failed experiment is a learning opportunity rather than a mistake, and that radical and controversial ideas should be tried. All these suggestions are challenging to put into practice in organisations that are not used to experimentation-based decision-making. Kohavi et al. note the challenges they faced at Microsoft, and describe efforts to raise awareness of the experimentation approach.

The final stage of the "stairway to heaven" model is detailed and analysed by Bosch [5]. The differences between traditional development and the continuous approach are analysed, showing that in the context of the new, continuous software development model, R&D is best described as an "innovation experiment system" approach where the development organisation constantly develops new hypotheses and tests them with certain groups of customers. This approach focuses on three phases:

pre-deployment, non-commercial deployment, and commercial deployment. The authors present a first systematisation of this so-called "innovation experiment system" adapted for software development for embedded systems. It is argued that aiming for an "innovation experiment system" is equally valid for embedded systems as it is in the case of cloud computing and Software-as-a-Service (SaaS), and that the process could be similar in both cases. That is, requirements should evolve in real time based on data collected from systems in actual use with customers.

Inspired by the ideas that define the last stage of the "stairway to heaven", we develop and propose the RIGHT model for Continuous Experimentation. In this model, experiments are derived from business strategies and aim to assess assumptions derived from those strategies, potentially invalidating or supporting the strategy. Previous works have explored the application of a framework for linking business goals and strategies to the software development activities (e.g., [2], [20]). However, those works have not considered the particular traits of an experiment system such as the one presented in this paper. The model presented also describes the platform infrastructure that is necessary to establish the whole experiment system. The Software Factory [8] can serve as infrastructure for the model proposed, as it is a software development laboratory well suited for continuous experimentation. In a previous article, in which we presented a study on creating minimum viable products [19] in the context of collaboration between industry and academia, we showed the Software Factory laboratory in relation to the Lean Startup approach and continuous experimentation. Some of the foundational ideas behind Software Factory with respect to continuous experimentation have been studied in the past, analysing, for instance, the establishment of laboratories specifically targeted for continuous development [21] and the impact of continuous integration in teaching software engineering.

The building blocks presented in this paper, although generalizable with certain limitations, are derived from a startup environment where the continuous experimentation approach is not only well suited but possibly the only viable option for companies to grow. Our work has similarities to the "Early Stage Startup Software Development Model" (ESSSDM) of Bosch et al. [6] which extends existing Lean Startup approaches offering more operational process support and better decision-making support for startup

companies. Specifically, ESSSDM provides guidance on when to move product ideas forward, when to abandon a product idea, and what techniques to use and when, while validating product ideas. Some of the many challenges faced when trying to establish a startup following the Lean Startup methodology are presented by May [18] with insights that we have considered for the present work.

3. Research approach

Our general research framework can be characterised as design science research [11], in which the purpose is to derive a technological rule which can be used in practice to achieve a desired outcome in a certain field of application [30]. The continuous experimentation model presented in this paper was first constructed based on the related work presented in the previous section as well the authors' experience. While a framework can be derived by purely analytic means, its validation requires grounding in empirical observations. For this reason, we conducted a holistic multiple case study [32] in the Software Factory laboratory at the Department of Computer Science, University of Helsinki, in which we matched the initial model to empirical observations and made subsequent adjustments to produce the final model. The model can still be considered tentative, pending further validation in other contexts. It is important to note that this study investigates how Continuous Experimentation can be carried out in a systematic way independently of the case projects' goals and the experiments carried out in them. Those experiments and their outcomes are treated as qualitative findings in the context of this study. In this section, we describe the case study context and the research process.

3.1. Context

The Software Factory is an educational platform for research and industry collaboration [8]. In Software Factory projects, teams of Master's-level students use contemporary tools and processes to deliver working software prototypes in close collaboration with industry partners. The goal of Software Factory activities is to provide students with means for applying their advanced software development skills in an environment with working life relevance and to deliver meaningful results for their customers [19].

During the case projects used in this study, two of the authors were involved as participant observers. The first author coordinated the case projects: started the projects, handled contractual and other administrative issues, followed up progress through direct interaction with the customer and student teams, ended the projects, handled project debriefing and coordinated the customer interviews. The third author also participated as an observer in several meetings where the customer and student teams collaborated. The researchers were involved in directing the experimentation design activities together with the customer, and students were not directly involved in these activities. However, the customer and students worked autonomously and were responsible for project management, technical decisions, and other issues related to the daily operations of the project.

3.1.1. Case Company 1

Tellybean Ltd.[1] is a small Finnish startup that develops a video calling solution for the home television set. During September 2012–December 2013 the company was a customer in three Software Factory projects with the aim of creating an infrastructure to support measurement and management of the architecture of their video calling service. Tellybean Ltd. aims at delivering a life-like video calling experience. Their value proposition – "the new home phone as a plug and play -experience" – is targeted at late adopter consumer customers who are separated from their families, e.g. due to migration into urban areas, global social connections, or overseas work. The company puts special emphasis on discovering and satisfying needs of the elderly, making ease of use the most important non-functional requirement of their product. The primary means for service differentiation in the marketplace are affordability, accessibility and ease of use. For the première commercial launch, and to establish the primary delivery channel of their product, the company aims at partnering with telecom operators. The company had made an initial in-house architecture and partial implementation during a pre-development phase prior to the Software Factory projects. A first project was conducted to extend the platform functionality of this implementation. A second project was conducted to

[1] `http://www.tellybean.com/`

validate concerns related to the satisfaction of operator requirements. After this project, a technical pivot was conducted, with major portions of the implementation being changed; the first two projects contributed to this decision. A third project was then conducted to extend the new implementation with new features related to the ability to manage software on already delivered products, enabling continuous delivery. The launch strategy can be described as an MVP launch with post-development adaptation. The three projects conducted with this company are connected to establishing a continuous experimentation process and building capabilities to deliver software variations on which experiments can be conducted. They also provided early evidence regarding the feasibility of the product for specific stakeholders, such as operator partners, developers, and release management.

3.1.2. Product

The Tellybean video calling service has the basic functionalities of a home phone: it allows making and receiving video calls and maintaining a contact list. The product is based on an Android OS set-top-box (STB) that can be plugged into a modern home TV. The company maintains a backend system for mediating calls to their correct respondents. While the server is responsible for routing the calls, the actual video call is performed as a peer to peer connection between STBs residing in the homes of Tellybean's customers.

The company played the role of a product owner in three Software Factory projects during September 2012–December 2013. The aim of the first two projects was to create new infrastructure for measuring and analysing usage of their product in its real environment. This information was important in order to establish the product's feasibility for operators and for architectural decisions regarding scalability, performance, and robustness. For the present research, the first two projects were used to validate the steps required to establish a continuous experimentation process. The third project at Software Factory delivered an automated system for managing and updating the STB software remotely. This project was used to investigate factors related to the architecture needs for continuous experimentation. Table 1 summarises the goals and motivations of the

Table 1: Scope of each of the three Tellybean projects at Software Factory.

	High-level goal	Motivation
Project 1	As an operator, I want to be able to see metrics for calls made by the video call product's customers.	. . . so that I can extract and analyse business critical information. . . . so that I can identify needs for maintenance of the product's technical architecture.
Project 2	As a Tellybean developer, I want to be sure that our product's system architecture is scalable and robust. As a Tellybean developer, I want to know technical weaknesses of the system. As a Tellybean developer, I want to receive suggestions for alternative technical architecture options.	. . . so that I know the limitations of the system. . . . so that I can predict needs for scalability of the platform. . . . so that I can consider future development options.
Project 3	As a technical manager, I want to be able to push an update to the Tellybean set-top-boxes with a single press of a button.	. . . so that I can deploy upgrades to the software on one or multiple set-top-boxes.

projects in detail. Each project had a 3–7 -person student team, a company representative accessible at all times, and spent effort in the range of 600 and 700 person-hours.

3.1.3. Project 1

The aim of Tellybean's first project at the Software Factory was to build means for measuring performance of their video calling product in its real environment. The goal was to develop a browser-based business analytics system. The team was also assigned to produce a back-end system for storing and managing data related to video calls, in order to satisfy operator monitoring requirements. The Software Factory project was carried out in seven weeks by a team of four Master's-level computer science students. Competencies required in the project were database design, application programming, and user interface design.

The backend system for capturing and processing data was built on the Java Enterprise Edition platform, utilising the Spring Open Source framework. The browser-based reporting system was built using JavaScript frameworks D3 and NVD3 to produce vivid and interactive reporting. A cache system of historical call data was implemented to ensure the performance of the system.

After the project had been completed, both students and the customer deemed that the product had been delivered according to the customer's requirements. Despite the fact that some of the foundational requirements changed during the project due to discoveries of new technological solutions, the customer indicated satisfaction with the end-product. During the project, communication between the customer and the team was frequent and flexible.

The first project constituted a first attempt at conducting continuous experimentation. The goal of the experiment was to gain information about the performance of the system architecture and its initial implementation. The experiment arose from operator needs to monitor call volumes and system load – a requirement that Tellybean's product developers deemed necessary to be able to partner with operators. It was clear that there existed a set of needs arising from operator requirements, but it was not clear how the information should be presented and what functionality was needed to analyse it. From

a research perspective, however, the exact details of the experiment were less important than the overall process of starting experimentation.

3.1.4. Project 2

The second project executed at Software Factory aimed at performing a system-wide stress test for the company's video calling service infrastructure. The Software Factory team of four Master's-level students produced a test tool for simulating very high call volumes. The tool was used to run several tests against Tellybean's existing call mediator server.

The test software suite included a tool for simulating video call traffic. The tool was implemented using the Python programming language. A browser-based visual reporting interface was also implemented to help analysis of test results. The reporting component was created using existing Javascript frameworks such as Highcharts.js and Underscore.js. Test data was stored in a MongoDB database to be utilised in analysis.

The purpose of the experiment was a counterpart to the experiment in the first project. Whereas the first project had focused on operator needs, the second focused on their implications for developers. The initial system architecture and many of the technical decisions had been questioned. The project aimed to provide evidence for decision-making when revisiting these initial choices.

The team found significant performance bottlenecks in Tellybean's existing proof-of-concept system and analysed their origins. Solutions for increasing operational capacity of the current live system were proposed and some of them were also implemented. Towards the end of the project, the customer suggested that a new proof-of-concept call mediating server should be proposed by the Software Factory team. The team delivered several suggestions for a new service architecture and composed a new call mediator server. For the purposes of this study, we consider the second experiment to be another round in the continuous experimentation cycle where findings from the first cycle resulted in a new set of questions to experiment on.

3.1.5. Project 3

For their third project at Software Factory, Tellybean aimed to create a centralised infrastructure for updating their video calling product's software components. The new remote software management system would allow the company to quickly deploy software updates to already delivered STBs. The functionality was business critical to the company and its channel partners: it allowed updating the software without having to travel on-location to each customer to update their STBs. The new instrument enabled the company to establish full control of their own software and hardware assets.

The project consisted of a team of five Master's-level computer science students. The team delivered a working prototype for rapid deployment of software updates. In this project, the need for a support system to deliver new features or software variations was addressed. We considered the architectural requirements for a continuous delivery system that would support continuous experimentation.

3.1.6. Case Company 2

Memory Trails Ltd. (Memory Trails) is a small Finnish startup that develops a well-being service which helps users define, track, and receive assistance with life goals. During May–July 2014, the company was a customer in a Software Factory project that aimed to develop a backend recommendation engine for the service, improve the front-end user experience, and to validate central assumptions in the service strategy. Memory Trails aims at delivering the service as an HTML5-based application which is optimised for tablets but also works on other devices with an HTML5-compatible browser. The service targets adults who wish to improve their quality of life and change patterns of behaviour to reach different kinds of life goals.

Whereas the projects with the first case company focused mostly on establishing a continuous experimentation process and building capabilities to deliver software variations for experimentation, the project with the second case company focused on some of the details of deriving experiments themselves. In particular, we sought to uncover how assumptions can be identified in initial product or service ideas. These assumptions are candidates for experiments of different kinds.

3.1.7. Project 4

Memory Trails provided an initial user interface and backend system prototype which demonstrated the general characteristics of the application from a user perspective. Users interact with photos which can be placed in different spatial patterns to depict emotional aspects of their goals. Users are guided by the application to arrange the photos as a map, showing the goal, potential steps towards it, and aspects that qualify the goals. For example, a life goal may be to travel around the world. Related photos could depict places to visit, moods to be experienced, items necessary for travel such as tickets, etc. The photos could be arranged, e.g., as a radial pattern with the central goal in the middle, and the related aspects around it, or as a time-line with the end goal to the right and intermediate steps preceding it.

In the project, two high-level assumptions were identified. The customer assumed that automatic, artificial intelligence-based processing in the backend could be used to automatically guide users towards their goals, providing triggers, motivation, and rewards on the way. Also, the customer assumed that the motivation for continued use of the application would come from interacting with the photo map. Since the automatic processing depended on the motivation assumption, the latter became the focus of experimentation in the project. The customer used versions of the application in user tests during which observation and interviews were used to investigate whether the assumption held. For the purposes of this study, we used the project to validate the link in our model between product vision, business model and strategy, and experiment steps.

3.2. Research process

The case study analysis was performed in order to ground the continuous experimentation model in empirical observations, not to understand or describe the projects themselves, nor to assess the business viability of the case companies. Therefore, we collected information that would help us understand the prerequisites for performing continuous experimentation, the associated constraints and challenges, and the logic of integrating experiment results into the business strategy and the development process.

We used four different sources of data in our analysis: (i) participant observation, (ii) analysis of project artefacts, (iii) group analysis sessions, and (iv) individual interviews. We subsequently discuss the details of the data collection and analysis.

During the projects, we observed the challenges the companies faced related to achieving the continuous experimentation system. At the end of each project, an in-depth debriefing session was conducted to gain retrospective insights into the choices made during the project, and the reasoning behind them. In addition to these sources, we interviewed three company representatives from Tellybean to understand their perception of the projects and to gain data which could be matched against our model. We also conducted a joint analysis session with the project team and two representatives from Memory Trails to further match insights on the experimentation process in their project with our model.

The debriefing sessions were conducted in a workshop-like manner, with one researcher leading the sessions and the project team, customer representatives, and any other project observer present. The sessions began with a short introduction by the leader, after which the attendees were asked to list events they considered important for the project. Attendees wrote down each event on a separate sticky note and placed them on a time-line which represented the duration of the project. As event-notes were created, clarifying discussion about their meaning and location on the time-line took place. When attendees could not think of any more events, they were asked to systematically recount the progress of the project using the time-line with events as a guide.

The interviews with customer representatives were conducted either in person on the customer's premises, online via video conferencing, or on the University of Helsinki's premises. The interviews were semi-structured thematic interviews, having a mixture of open-ended and closed questions. This interview technique allows participants to freely discuss issues related to a focal theme. Thematic interviews have the advantage that they provide opportunities to discover information that researchers cannot anticipate and that would not be covered by more narrowly defined, closed questions. While they may result in the discussion straying away from the focal theme, this is not a problem in

practice since the interviewer can direct the participant back to the theme and irrelevant information can be ignored in the analysis.

A minimum of two researchers were present in the interviews to ensure that relevant information was correctly extracted. All participating researchers took notes during the interviews, and notes were compared after the interviews to ensure consistency. In the interviews, company representatives were first asked to recount their perception of their company, its goals, and its mode of operation before the three projects. Then, they were asked to consider what each project had accomplished in terms of software outcomes, learned information, and implications for the goals and mode of operation of the company. Finally, they were asked to reflect on how the company operated at the time of the interview and how they viewed the development process, especially in terms of incorporating market feedback into decision-making.

During analysis, the project data were examined for information relevant to the research question. We categorised the pieces of evidence according to whether they related to the Continuous Experimentation process or to the infrastructure. We sought to group the observations made and understanding gained during the projects with evidence from the retrospective sessions and interviews so that the evidence was triangulated and thus strengthened. Such groups of triangulated evidence was then matched with our initial model, which was similar to the sequence shown in Figure 1, and included the build-measure-learn cycle for the process, and a data repository, analysis tools, and continuous delivery system as infrastructure components. We adjusted the model and introduced new process steps and infrastructure components that supported the need implied by the evidence. We strived for minimal models, and when more than one need could be fulfilled with a single step or component, we did not introduce more steps or components. When all the evidence had been considered, we evaluated the result as a whole and made some adjustments and simplifications based on our understanding and judgement.

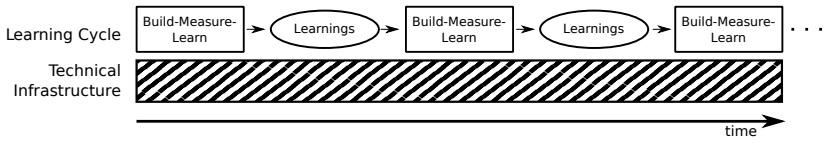

Figure 1: Sequence of RIGHT process blocks.

4. Results

In this section, we first describe our proposed model for continuous experimentation, and then report on the insights gained from the multiple case study and how they inform the different parts of the model.

4.1. The RIGHT model for Continuous Experimentation

By continuous experimentation, we refer to a software development approach that is based on field experiments with relevant stakeholders, typically customers or users, but potentially also with other stakeholders such as investors, third-party developers, or software ecosystem partners. The model consists of repeated Build-Measure-Learn blocks, supported by an infrastructure, as shown in Figure 1. Each Build-Measure-Learn block results in learnings which are used as input for the next block. Conceptually, the model can also be thought to apply not only to software development, but also to design and development of software-intensive products and services. In some cases, experimentation using this model may require little or no development of software.

The Build-Measure-Learn blocks structure the activity of conducting experiments, and connect product vision, business strategy, and technological product development through experimentation. In other words, the requirements, design, implementation, testing, deployment, and maintenance phases of software development are integrated and aligned by empirical information gained through experimentation. The model can be considered a vehicle for incremental innovation as defined by Henderson and Clark [10], but the model itself, as well as the transition to continuous experimentation in general, can be considered radical, architectural innovations that require significant new organisational capabilities.

4.1.1. The RIGHT process model for Continuous Experimentation

Figure 2 expands the Build-Measure-Learn blocks and describes the RIGHT process model for Continuous Experimentation. A general vision of the product or service is assumed to exist. Following the Lean Startup methodology [26], this vision is fairly stable and is based on knowledge and beliefs held by the entrepreneur. The vision is connected to the business model and strategy, which is a description of how to execute the vision. The business model and strategy are more flexible than the vision, and consist of multiple assumptions regarding the actions required to bring a product or service to market that fulfils the vision and is sustainably profitable. However, each assumption has inherent uncertainties. In order to reduce the uncertainties, we propose to conduct experiments. An experiment operationalises the assumption and states a hypothesis that can be subjected to experimental testing in order to gain knowledge regarding the assumption. The highest-priority hypotheses are selected first. Once a hypothesis is formulated, two parallel activities can occur. The hypothesis can optionally be used to implement and deploy a Minimum Viable Product (MVP) or Minimum Viable Feature (MVF), which is used in the experiment and has the necessary instrumentation. Simultaneously, an experiment is designed to test the hypothesis. The experiment is then executed and data from the MVP/MVF are collected in accordance with the experimental design. The resulting data are analysed, concluding the experimental activities.

Once the experiment has been conducted and analysis performed, the analysis results are used on the strategy level to support decision-making. Again following Lean Startup terminology, the decision can be to either "pivot" or "persevere" [26], but a third alternative is also possible: to change assumptions in the light of new information. If the experiment has given support to the hypothesis, and thus the assumption on the strategy level, a full product or feature is developed or optimised, and deployed. The strategic decision in this case is to persevere with the chosen strategy. If, on the other hand, the hypothesis was falsified, invalidating the assumption on the strategy level, the decision is to pivot and alter the strategy by considering the implications of the assumption being false. Alternatively, the tested assumption could be changed, but not

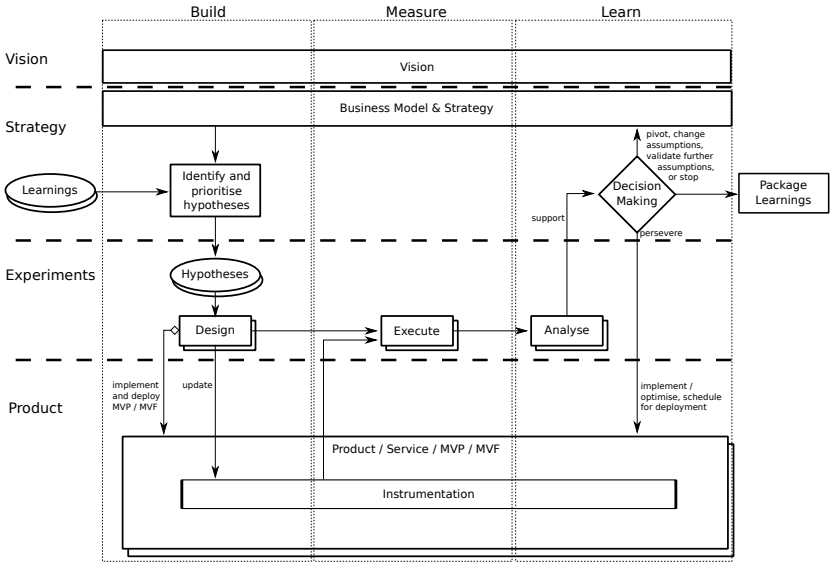

Figure 2: The RIGHT process model for Continuous Experimentation.

completely rejected, depending on what the experiment was designed to test and what the results were.

4.1.2. The RIGHT infrastructure architecture for Continuous Experimentation

To support conducting such experiments, an infrastructure for continuous experimentation is needed. Figure 3 sketches the RIGHT infrastructure architecture for Continuous Experimentation, with roles and associated tasks, the technical infrastructure, and information artefacts. The roles indicated here will be instantiated in different ways depending on the type of company in question. In a small company, such as a startup, a small number of persons will handle the different roles and one person may have more than one role. In a large company, the roles are handled by multiple teams. Seven roles are defined to handle four classes of tasks. A business analyst and a product owner, or a product management team, together handle the creation and iterative updating of the strategic roadmap. In order to do so, they consult existing experimental plans, results, and learnings, which reside in a back-end system. As plans and results accumulate and are stored, they may be reused in further development of the roadmap. The business

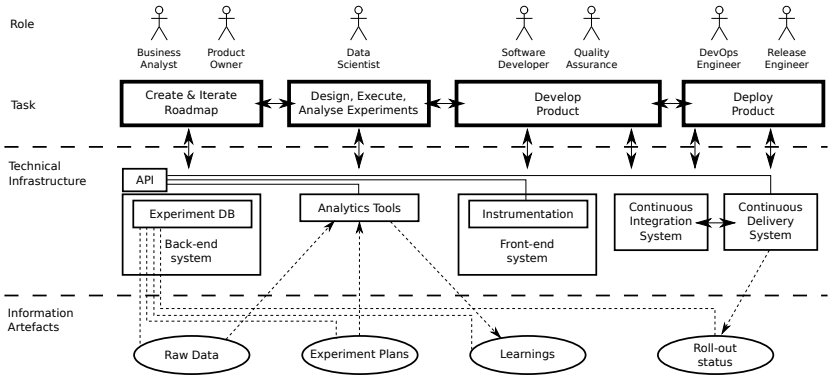

Figure 3: The RIGHT infrastructure architecture for Continuous Experimentation.

analyst and product owner work with a data scientist role, which is usually a team with diverse skills, to communicate the assumptions of the roadmap and map the areas of uncertainty which need to be tested.

The data scientist designs, executes, and analyses experiments. A variety of tools are used for this purpose, which access raw data in the back-end system. Conceptually, raw data and experiment plans are retrieved, analysis performed, and results produced in the form of learnings, which are stored back into the back-end system.

The data analyst also communicates with a developer and quality assurance role. These roles handle the development of MVPs, MVFs, and the final product. They first work with the data analyst to produce proper instrumentation into the front-end system, which is the part of the software which is delivered or visible to the user. In the case of a persevere-decision, they work to fully develop or optimise the feature and submit it for deployment into production. MVPs, MVFs, and final products are deployed to users after first going through the continuous integration and continuous delivery systems. A DevOps engineer acts as the mediator between the development team and operations, and a release engineer may oversee and manage the releases currently in production. Importantly, the continuous delivery system provides information on software roll-out status, allowing other roles to monitor the experiment execution and, e.g., gain an understanding of the conditions under which the software was deployed to users and of the sample characteristics and response rate of the experiment. Cross-cutting concerns

such as User Experience may require additional roles working with several of the roles mentioned here. To simplify the figure, we have omitted the various roles that relate to operations, such as site reliability engineer, etc. Also, we have omitted a full elaboration of which information artefacts should be visible to which roles. In general, we assume that is is beneficial to visualise the state of the continuous experimentation system for all roles.

The back-end system consists of an experiment database which, conceptually, stores raw data collected from the software instrumentation, experiment plans – which include programmatic features of sample selection and other logic needed to conduct the experiment – and experiment results. The back-end system and the database are accessible through an API. Here, these parts should be understood as conceptual; an actual system likely consists of multiple APIs, databases, servers, etc. The experiment database enables a product architecture where deployed software is configured for experiments at run-time. Thus it is not always required that a new version of the software or the accompanying instrumentation is shipped to users prior to an experiment; the experimental capability can be built into the shipped software as a configurable variation scheme. The shipped software fetches configuration parameters for new experiments, reconfigures itself, and sends back the resulting measurement data, eliminating the need to perform the Develop Product and Deploy Product tasks. For larger changes, a new software version may be required, and the full set of tasks performed.

4.2. Model instantiations and lessons learned

In this subsection, we describe how the RIGHT models were instantiated in the four projects, and we describe the lessons learned. We include illustrative examples from our interview data. We note that the model was initially quite simple, similar to the sequence described in Figure 1 with a build-measure-learn cycle, a data repository, analysis tools, and continuous delivery system. We also note that not all parts of the models were instantiated in all projects. We assume that this will be the case in other projects as well. In the first two projects, we focused on problem validation: developing an understanding of the needs in real situations that a model for continuous experimentation should address. In the two latter projects, we already had most of the model in place and

focused more on validating our solution, using detailed findings from the projects in order to adjust the model.

Each of the four case projects relate to different aspects of continuous experimentation. The case findings support the need for systematic integration of all levels of software product and service development, especially when the context is rapid new product and service development. The key issue is to develop a product that customers will buy, given tight financial constraints. Startup companies operate in volatile markets and under high uncertainty. They may have to do several quick changes as they get feedback from the market. The challenge is to reach product-market fit before running out of money.

> "You have to be flexible because of money, time and technology constraints. The biggest question for us has been how to best use resources we have to achieve our vision. In a startup, you are time-constrained because you have a very limited amount of money. So you need to use that time and money very carefully." (Tellybean founder)

When making changes in the direction of the company, it is necessary to base decisions on sound evidence rather than guesswork. However, we found that it is typically not the product or service vision that needs to change. The change should rather concern the strategy by which the vision is implemented, including the features that should be implemented, their design, and the technological platform on which the implementation is based. For example, although Tellybean has had to adapt several times, the main vision of the company has not changed.

> "The vision has stayed the same: lifelike video calling on your TV. It is very simple; everyone in the company knows it. The TV part doesn't change, but the business environment is changing. The technology – the hardware and software – is changing all the time." (Tellybean founder)

> "We had to pivot when it comes to technology and prioritising features. But the main offering is still the same: it's the new home phone and it connects to your TV. That hasn't changed. I see the pivots more like springboards to

> the next level. For example, we made a tablet version to [gain a distributor partner]." (Tellybean CTO)

Also, although an experiment design is, at best, self-evident when viewed in hindsight, developing one based on the information available in actual software projects, especially new product or service development, is not an easy task. There are multiple possibilities for what to experiment on, and it is not obvious how to choose the first experiment or each next experiment after that. Our case projects showed that initiating the continuous experimentation process is a significant task in its own right and involves much learning. This strengthens the notion that a basic and uncomplicated model to guide the process in the right direction is needed.

4.2.1. Project 1

In the first project, the new business analytics instrument allowed Tellybean to yield insights on their system's statistics, providing the company a means for feedback. They could gain a near-real-time view on call related activities, yielding business critical information for deeper analysis. The presence of the call data could be used as input for informed decisions. It also allowed learning about service quality and identifying customer call behaviour patterns. Based on the customer's comments, such information would be crucial for decision-making regarding the scaling of the platform. Excess capacity could thus be avoided and the system would be more profitable to operate while still maintaining a good service level for end users. The primary reason for wanting to demonstrate such capabilities was the need to satisfy operator needs. To convince operators to become channel partners, the ability to respond to fluctuations in call volumes was identified as critical. Potential investors would be more inclined to invest in a company that could convince channel operators of the technical viability of the service.

> "There were benefits in terms of learning. We were able to show things to investors and other stakeholders. We could show them examples of metric data even if it was just screenshots." (Tellybean CTO)

The high-level goal of the first project could be considered as defining a business hypothesis to test the business model from the viewpoint of the operators. The project delivered the needed metrics as well as a tool-supported infrastructure to gather the necessary data. These results could be used to set up an experiment to test the business hypotheses.

Table 2 shows the parts of our model that were instantiated in Project 1. The project instantiated a few basic elements of the RIGHT process model. The chosen business model and strategy was to offer the video calling service through operator partnerships. In order for the strategy to be successful, the company needed to demonstrate the feasibility of the service in terms of operator needs and requirements. This demonstration was to operators themselves but also to other stakeholders, such as investors, who assessed the business model and strategy. The hypothesis to test was not very precisely defined in the project, but could be summarised as "operators will require system performance management analysis tools in order to enter a partnership". The experiment, which was obviously not a controlled one but rather conducted as part of investor and operator negotiations, used the analytics instrument developed in the project to assess whether the assumption was correct, thus instantiating an MVF, and making a rudimentary experiment execution and analysis. Based on this information, some decisions were made: to start investigating alternative architectures and product implementation strategies.

4.2.2. Project 2

In the second project, Tellybean was able to learn the limitations of the current proof-of-concept system and its architecture. An alternative call mediator server and an alternative architecture for the system were very important for the future development of the service. The lessons learned in the second project, combined with the results of the first, prompted them to pivot heavily regarding the technology, architectural solutions, and development methodology.

> "The Software Factory project [...] put us on the path of 'Lego software development', building software out of off-the-shelf, pluggable components. It got us thinking about what else we should be doing differently. [...] We

Table 2: Model instantiations in Project 1.

Process model instantiation	
Vision	Video calling in the home
Business model and strategy	Offer video calling through operator partnerships (+ assumptions about architecture and product implementation strategies)
Hypotheses	"Operators will require performance management analysis tools in order to enter a partnership"
Design, execute, analyse	Rudimentary
MVF	Analytics instrument
Decision making	Start architectural pivot (continued in Project 2) Start product implementation strategy pivot (continued in Project 2) Validate further assumptions (regarding architecture and product implementation)
Infrastructure model instantiation (only applicable parts)	
Roles	Business analyst, product owner (played by company leadership), software developer (played by Software Factory students)
Technical Infrastructure	Analytics Tools (MVF developed in project)
Information Artefacts	Learnings (not formally documented in project)

were thinking about making our own hardware. We had a lot of risk and high expenses. Now we have moved to existing available hardware. Instead of a client application approach, we are using a web-based platform. This expands the possible reach of our offering. We are also looking at other platforms. For example, Samsung just released a new SDK for Smart TVs." (Tellybean founder)

"Choosing the right Android-based technology platform has really sped things up a lot. We initially tried to do the whole technology stack from hardware to application. The trick is to find your segment in the technology stack, work there, and source the rest from outside. We have explored several Android-based options, some of which were way too expensive. Now we have started to find ways of doing things that give us the least amount of problems. But one really important thing is that a year ago, there were no Android devices like this. Now there are devices that can do everything we need. So the situation has changed a lot." (Tellybean CTO)

The high-level goals of the second project could be considered as defining and testing a solution hypothesis that addresses the feasibility of the proposed hardware-software solution. The project delivered an evaluation of the technical solution as well as improvement proposals. The analysis showed that the initial architecture and product implementation strategy were too resource-consuming to carry out fully. The results were used by the company to modify their strategy. Instead of implementing the hardware themselves, they opted for a strategy where they would build on top of generic hardware platforms and thus shorten time-to-market and development costs. Table 3 shows the model instantiations in Project 2.

4.2.3. Project 3

In the third project, the capability for continuous deployment was developed. The STBs could be updated remotely, allowing new features to be pushed to customers at very low cost and with little effort. The implications of this capability are that the company is able to react to changes in their technological solution space by updating operating

Table 3: Model instantiations in Project 2.

Process model instantiation	
Vision	Video calling in the home
Business model and strategy	Offer video calling through operator partnerships (+ assumptions about architecture and product implementation strategies)
Hypotheses	"Product should be developed as custom hardware-software codesign" and "Architecture should be based on Enterprise Java technology and be independent of TV set (which acts only as display)"
Design, execute, analyse	Prototype implementation; evaluate current solution proposal
MVF	Alternative call mediator server; alternative system architecture
Decision making	Architectural pivot (Android-based COTS hardware and OS) Product implementation strategy pivot (do not develop custom hardware)
Infrastructure model instantiation (only applicable parts)	
Roles	Business analyst, product owner (played by company leadership), software developer (played by Software Factory students)
Technical Infrastructure	Analytics Tools (from previous project)
Information Artefacts	Learnings (not formally documented in project)

system and application software, and to emerging customer needs by deploying new features and testing feature variants continuously.

The high-level goals of the third project could be considered as developing a capability that allows for automating the continuous deployment process. The prerequisite for this is a steady and controlled pace of development where the focus is on managing the amount of work items that are open concurrently in order to limit complexity. At Tellybean, this is known as the concept of one-piece flow.

> "The one-piece flow means productisation. In development, it means you finish one thing before moving on to the next. It's a bit of a luxury in development, but since we have a small team, it's possible. On the business side, the most important thing has been to use visual aids for business development and for prioritising. In the future we might try to manage multiple-piece flows." (Tellybean founder)

The third project instantiated parts of our infrastructure architecture model, shown in Table 4. In particular, it focused on the role of a continuous delivery system in relation to the tasks that need to be carried out for continuous experimentation, meaning that top and rightmost parts of Figure 3 were instantiated, as detailed in the table.

4.2.4. Project 4

In the fourth project, it was initially difficult to identify what the customers considered to be the main assumptions. However, once the main assumptions became clear, it was possible to focus on validating them. This highlights the finding that although it is straightforward in theory to assume that hypotheses should be derived from the business model and strategy, it may not be straightforward in practice. In new product and service development, the business model and strategy is not finished, and, especially in the early cycles of experimentation, it may be necessary to try several alternatives and spend effort on modelling assumptions until a good set of hypotheses is obtained. We therefore found it useful to separate the identification and prioritisation of hypotheses on the strategy level from the detailed formulation of hypotheses and experiment design on the experiment level. Table 5 shows the instantiated model parts in Project 4. We

Table 4: Model instantiations in Project 3.

Process model instantiation	
Vision	Video calling in the home
Business model and strategy	Offer video calling through operator partnerships (+ assumptions about architecture and product implementation strategies)
Hypotheses	"Capability for automatic continuous deployment is needed for incremental product development and delivery"
Design, execute, analyse	Project focused on instantiating parts of infrastructure architecture model and did not include a product experiment
MVF	Prototype for rapid deployment of software updates
Decision making	Persevere
Infrastructure model instantiation (only applicable parts)	
Roles	Business analyst, product owner (played by company leadership), software developer (played by Software Factory students), DevOps engineer, release engineer (played by company CTO and other technical representatives; also represented by user stories with tasks for these roles)
Technical infrastructure	Continuous integration system, continuous delivery System (MVF developed in project)
Information artefacts	Roll-out status

note that some of these parts were introduced into the model because of our findings from Project 4.

In this project, there were two assumptions: that interaction with the photo map would retain users, and that an automated process of guiding users towards goals was feasible. The assumption that continued use of the application would come from interacting with the photo map was shown to be incorrect. Users would initially create the map, but would not spend much time interacting with it – by, e.g., adding or changing photos, rearranging the map, adding photo annotations, etc. Instead, users reported a desire for connecting with other users to share maps and discuss life goals. Also, they expressed a willingness to connect with professional or semi-professional coaches to get help with implementing their life goals. The social aspect of the service had been overlooked. Whether this was due to familiarity with existing social media applications was left uninvestigated. In any case, the assumption was invalidated and as a result, the assumptions regarding automated features for guiding users towards goals were also invalidated. The investigation indicated that users were motivated by the potential for interaction with other users, and that these interactions should include the process of motivating them to reach goals. It is important to note that the two hypotheses could be invalidated because they were dependent. The process of identifying and prioritising hypotheses separately from detailed formulation of hypotheses and experiment design makes it possible to choose the order of experiments in a way that gains the maximum amount of information with the minimum number of experiments. Testing the most fundamental assumptions – the ones on which most other assumptions rely – first, allows the possibility of eliminating other assumptions with no additional effort.

The fourth project also revealed challenges involved with instrumenting the application for data collection. It was difficult to separate the process of continuous experimentation from the technical prerequisites for instrumentation. In many cases, substantial investments into technical infrastructure may be needed before experiments can be carried out. These findings led to the roles, the high-level description of the technical infrastructure, and the information artefacts in the infrastructure architecture (see Figure 3).

Table 5: Model instantiations in Project 4.

Process model instantiation	
Vision	Well-being service for defining, tracking, and receiving assistance with life goals
Business model and strategy	Product and service recommendations, automated recommendation engine for motivating progress towards goals
Hypotheses	"Motivation for continued use comes from interacting with photo map" and "Automatic recommendation engine will automatically guide users to reach goals" (depends on first hypothesis)
Design, execute, analyse	User tests with observation and interviews
MVF	HTML5-based, tablet-optimised application
Decision making	Product implementation strategy pivot (focus on social interaction rather than automated recommendations)
Infrastructure model instantiation (only applicable parts)	
Roles	Business analyst, product owner (played by company leadership), software developer (played by Software Factory students)
Technical infrastructure	Instrumentation, Front-end system
Information artefacts	Learnings

However, many experiments are also possible without advanced instrumentation. The fourth project indicates that experiments may typically be large, or target high-level questions, in the beginning of the product or service development cycle. They may address questions and assumptions which are central to the whole product or service concept. Later stages of experimentation may address more detailed aspects, and may be considered optimisation of an existing product or service.

5. Discussion

The continuous experimentation model developed in the previous section can be seen as a general description. Many variations are possible. For instance, experiments may be deployed to selected customers in a special test environment, and several experiments may be run in parallel. A special test environment may be needed particularly in business-to-business markets, where the implications of feature changes are broad and there may be reluctance towards having new features at all. The length of the test cycle may thus have to be longer in business-to-business markets. Direct deployment could be more suitable for consumer markets, but we note that the attitude towards continuous experimentation is likely to change as both business and consumer customers become accustomed to it.

Each project could have instantiated the RIGHT models in different ways. In the first project, the experiment could have been carried out using mockup screens to validate what metric data, visualisation, and analysis tools would have been sufficient to convince the stakeholders. However, this would have been detrimental since it would not have revealed the shortcomings in the initial architecture and implementation strategy. Although the design of the experiment left much to be desired, carrying it out using a real, programmed prototype system made it possible to discover the need to reconsider some of the previous strategy choices.

In the second project, the learnings could have been better used to define a more precise set of hypotheses after a careful analysis of the shortcomings of the previous system architecture. However, this was not necessary since the purpose was not a point-by-point comparison but rather an either-or comparison between one general approach

and another. This highlights an important notion regarding continuous experimentation: it only seeks to produce enough information for a decision to be made correctly.

In the third project, only the capability for continuous delivery was instantiated. The project could also have addressed the components that are necessary to carry out actual experiments. Due to project time constraints, this was left uninvestigated in the third project, but was considered in the fourth project instead. In that project, one cycle of the full RIGHT process model was carried out, and the software was instrumented for experimentation although using ready-made services such as Google Analytics.

While our ultimate aim is for our models to cover the entire breadth of continuous experimentation, we assume that not all real-life projects will need to instantiate all parts. For instance, experiments can be conducted without an MVP, especially in an early stage of product development. It may also not be necessary in all cases to have a heavy infrastructure for the experimentation – this becomes relevant if experimentation is conducted in very large volumes or when the purpose is to maintain a set of experiments that are run continuously to collect trend information while the product is incrementally changed.

In addition to the project-specific observations, we consider some more general concerns. Having several experiments run in parallel presents a particular challenge. The difficulty of interpreting online experiments has been convincingly demonstrated by Kohavi et al. [16]. Statistical interactions between experiments should be considered in order to assess the trustworthiness of the experiments. For this reason, it is important to coordinate the design and execution of experiments so that correct inferences are drawn. More generally, the issue of validity becomes important when the entire R&D organisation is experiment-driven. Incorrectly designed or implemented experiments may lead to critical errors in decision-making. Threats to validity can also stem from a failure to consider ethical aspects of experiments. Not only may unethical experiments damage company reputation, but they may cause respondents to knowingly or unconsciously bias the experimental results, leading to errors in decision-making.

Other challenges include the difficulty of prioritising where to start: which assumption should be tested first. In Project 4, we identified a dependency between assumptions regarding the backend recommendation logic and the assumption of what motivates users

to keep using the application. By invalidating the latter, we automatically invalidated the first assumption. This highlights the importance of identifying critical assumptions, as testing them first may save several unneeded experiments. We see a need for further research into this area. Also, in hardware-software co-design, illustrated by the first three projects, setting up the experimental cycle quickly is a major challenge due to both the longer release cycle of hardware and the potential synchronisation problems between hardware and software development schedules. Based on the findings presented in this paper, it may be beneficial to test a few strategic technical assumptions first, such as the viability of a certain hardware-software platform. As our case demonstrates, choosing the correct platform early can have a significant impact on the ability to proceed to actual service development.

A further set of challenges have to do with the model of sales and supplier networks. Essentially all companies are dependent on a network of suppliers and sales channels. It may be necessary to extend the model presented here to take into account the capabilities particularly of hardware suppliers to supply the needed components in a timely fashion and with the needed flexibility to programmatically vary behavioural parameters in these components. Also, when the company is not selling its products directly to end users, several levels of intermediaries may interfere with the possibilities to collect data directly from field use. If a sales partner cannot grant access to end users, other means of reaching the audience are needed. We envision using early-access and beta-test programs for this purpose, a practice that is commonly used in the computer gaming industry. Other models are possible, and there is an opening for further research in this area.

In some cases, an experimental approach may not be suitable at all. For example, certain kinds of life-critical software or software that is used in environments where experimentation is prohibitively expensive, may preclude the use of experiments as a method of validation. However, it is not clear how to determine the suitability of an experimental approach in specific situations, and research on this topic could yield valuable guidelines on when to apply the model presented here.

Another question is whether continuous delivery is a strictly necessary precondition for continuous experimentation. In the beginning of the product development cycle,

experimentation must occur before much software is written at all. At that stage, continuous delivery may not be necessary. Also, not all experiments require new software to be delivered to users. While a continuous delivery system may exist, the software itself may be architected for variability so that it can reconfigure itself at run-time. In such cases, no new version of the software needs to be delivered for new experiments to run. However, not all experiments are possible even with a very flexible architecture that allows for run-time reconfiguration. Continuous delivery is a good vehicle for delivering experiments to users and to ensure quality in the development process. The model presented here is based on iterative, evolutive optimisation of product features and an incremental model of innovation. To carry out revolutionary innovation, the process needs to be extended with other means of discovering customer value. These may profoundly invalidate the business model or strategy, and may even have an impact on the overall vision.

Finally, experimentation may be conducted with several kinds of stakeholders. Apart from customers and end users, experiments could be directed towards investors, suppliers, sales channels, or distributors. Companies whose product is itself a development platform may want to conduct experiments with developers in their platform ecosystem to optimise the developer experience [9] of their tools, methods, and processes. These experiments may require other kinds of experimental artefacts than the MVP/MVF, including, e.g., processes, APIs, and documentation. Research on the types of experimental artefacts and associated experimental designs could lead to fruitful results for such application areas. Also, an open question is who should primarily lead or conduct the experimentation, especially when the development organisation is separate from the customer organisation. Some training may be needed for customers in order to ensure that they can interact with the continuous experimentation process running in the development organisation. Similarly, the development team may need additional training to be able to interact with the customer to derive assumptions, plan experiments, and report results for subsequent decision-making. Another possibility is to introduce a mediating role which connects the customer and development organisations. More generally, increasing the capability to perform experimentation and continuous software engineering requires consideration of human factors in software development teams [23].

Further research is needed to determine how the experimental process works across organisational borders, whether within or outside a single company.

A particular limitation of this study is the use of relatively short projects with student participants. Students carried out the technical software development and analysis tasks in the projects, while the researchers handled tasks related to identification of assumptions, generation of hypotheses, and higher-level planning tasks together with customer representatives. While it is reasonable to expect that professional software developers would have reached a different level of quality and rigour in the technical tasks, we consider it likely that the findings are applicable beyond student projects since the focus of this paper is not on the technical implementation but on the integration of experiment results in the product development cycle and the software development process. The length of the projects means that at most one experimental cycle could be carried out in a single project. Thus the first case company completed three, and the second case company one experimental cycle. In a real setting, multiple experimentation rounds would be carried out over an extended period of time, proceeding from experiments addressing the most important assumptions with the highest impact towards increasing detail and optimisation. The findings of this study should be considered to apply mostly in the early stages of experimentation.

6. Conclusions

Companies are increasingly transitioning their traditional research and product development functions towards continuous experiment systems [12]. Integrating field experiments with product development on business and technical levels is an emerging challenge. There are reports of many companies successfully conducting online experiments, but there is a lack of a systematic framework model for describing how such experiments should be carried out and used systematically in product development. Empirical studies on the topic of continuous experimentation in software product development is a fruitful ground for further research. Software companies would benefit from clear guidelines on when and how to apply continuous experimentation in the design and development of software-intensive products and services.

In this paper, we match a model for Continuous Experimentation based on analysis of previous research against a multiple case study in the Software Factory laboratory at the University of Helsinki. The model describes the experimentation process, in which assumptions for product and business development are derived from the business strategy, systematically tested, and the results used to inform further development of the strategy and product. The infrastructure architecture for supporting the model takes into account the roles, tasks, technical infrastructure, and information artefacts needed to run large-scale continuous experiments.

A system for continuous experimentation requires the ability to release minimum viable products or features with suitable instrumentation, design and manage experiment plans, link experiment results with a product roadmap, and manage a flexible business strategy. There are several critical success factors for such a system. The organisation must be able to properly and rapidly design experiments, perform advanced instrumentation of software to collect, analyse, and store relevant data, and integrate experiment results in both the product development cycle and the software development process. Feedback loops must exist through which relevant information is fed back from experiments into several parts of the organisation. A proper understanding of what to test and why must exist, and the organisation needs a workforce with the ability to collect and analyse qualitative and quantitative data. Also, it is crucial that the organisation has the ability to properly define decision criteria and act on data-driven decisions.

In future work, we expect the model to be expanded as more use cases arise in the field. Domain-specific variants of the model may also be needed. Furthermore, there are many particular questions with regard to the individual parts of the model. Some specific areas include (i) how to prioritise assumptions and select which assumptions to test first; (ii) how to assess validity and determine how far experimental results can be trusted – especially how to ensure that experiments are trustworthy when running potentially thousands of them in parallel; (iii) how to select proper experimental methods for different levels of product or service maturity; and (iv) how to build a back-end system for continuous experimentation that can scale to the needs of very large deployments, and can facilitate and even partially automate the creation of experimental plans. Particular questions regarding automation include which parts of the model could be automated or

supported through automation. Another question is how quickly a Build-Measure-Learn block can be executed, and what the performance impact of the model is on the software development process.

Acknowledgements

This work was supported by Tekes – the Finnish Funding Agency for Technology and Innovation, as part of the N4S Program of DIGILE (Finnish Strategic Centre for Science, Technology and Innovation in the field of ICT and digital business).

References

[1] Rob J. Adams, Bradee Evans, and Joel Brandt, *Creating Small Products at a Big Company: Adobe's Pipeline Innovation Process*, CHI'13 Extended Abstracts on Human Factors in Computing Systems, 2013, pp. 2331–2332.

[2] V Basili, J Heidrich, M Lindvall, J Münch, M Regardie, D Rombach, C Seaman, and A Trendowicz, *GQM$^+$Strategies: A comprehensive methodology for aligning business strategies with software measurement*, Proceedings of the DASMA Software Metric Congress (MetriKon 2007): Magdeburger Schriften zum Empirischen Software Engineering, 2007, pp. 253–266.

[3] V. Basili, R. Selby, and D. Hutchens, *Experimentation in Software Engineering*, IEEE Transactions on Software Engineering **12** (1986), no. 7, 733–743.

[4] Steve Blank, *The Four Steps to the Epiphany: Successful Strategies for Products that Win*, 2nd ed., K&S Ranch, 2013.

[5] Jan Bosch, *Building Products as Innovation Experiment Systems*, Software Business, 2012, pp. 27–39.

[6] Jan Bosch, Helena Holmström Olsson, Jens Björk, and Jens Ljungblad, *The Early Stage Software Startup Development Model: A Framework for Operationalizing Lean Principles in Software Startups*, Lean Enterprise Software and Systems, 2013, pp. 1–15.

[7] U. Eklund and J. Bosch, *Architecture for Large-Scale Innovation Experiment Systems*, Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012, pp. 244–248.

[8] F. Fagerholm, N. Oza, and J. Münch, *A platform for teaching applied distributed software development: The ongoing journey of the Helsinki software factory*, 3rd International Workshop on Collaborative Teaching of Globally Distributed Software Development (CTGDSD), 2013, pp. 1–5.

[9] Fabian Fagerholm and Jürgen Munch, *Developer Experience: Concept and Definition*, International Conference on Software and System Process, 2012, pp. 73–77.

[10] Rebecca M. Henderson and Kim B. Clark, *Architectural Innovation: The Reconfiguration of Existing Product Technologies and the Failure of Established Firms*, Administrative Science Quarterly **35** (1990), no. 1, 9–30.

[11] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram, *Design Science in Information Systems Research*, MIS Quarterly **28** (2004), no. 1, 75–105.

[12] Helena Holmström Olsson, Hiva Alahyari, and Jan Bosch, *Climbing the "Stairway to Heaven" – A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software*, 39th EUROMICRO Conference on Software Engineering and Advanced Applications (2012), 392–399.

[13] Helena Holmström Olsson and Jan Bosch, *The HYPEX Model: From Opinions to Data-Driven Software Development*, Continuous Software Engineering, 2014, pp. 155–164.

[14] Natalia Juristo and Ana M. Moreno, *Basics of Software Engineering Experimentation*, Springer, 2001.

[15] Ron Kohavi, Thomas Crook, and Roger Longbotham, *Online Experimentation at Microsoft*, Third Workshop on Data Mining Case Studies and Practice Prize, 2009.

[16] Ron Kohavi, Alex Deng, Brian Frasca, Roger Longbotham, Toby Walker, and Ya Xu, *Trustworthy Online Controlled Experiments: Five Puzzling Outcomes Explained*, Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2012, pp. 786–794.

[17] Ron Kohavi, Alex Deng, Brian Frasca, Toby Walker, Ya Xu, and Ni Pohlmann, *Online Controlled Experiments at Large Scale*, Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13), 2013, pp. 1168–1176.

[18] Beverly May, *Applying Lean Startup: An Experience Report – Lean & Lean UX by a UX Veteran: Lessons Learned in Creating & Launching a Complex Consumer App*, Agile Conference (AGILE) 2012, 2012, pp. 141–147.

[19] Jürgen Münch, Fabian Fagerholm, Patrik Johnson, Janne Pirttilahti, Juha Torkkel, and Janne Järvinen, *Creating Minimum Viable Products in Industry-Academia Collaborations*, Proceedings of the Lean Enterprise Software and Systems Conference (LESS 2013, Galway, Ireland, December 1-4), 2013, pp. 137–151.

[20] Jürgen Münch, Fabian Fagerholm, Petri Kettunen, Max Pagels, and Jari Partanen, *Experiences and Insights from Applying GQM+Strategies in a Systems Product Development Organisation*, Proceedings of the 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2013), 2013.

[21] Jim Nieters and Amit Pande, *Rapid Design Labs: A Tool to Turbocharge Design-led Innovation*, Interactions (2012), 72–77.

[22] Taiichi Ōno, *Toyota production system: beyond large-scale production*, Productivity press, 1988.

[23] Efi Papatheocharous, Marios Belk, Jaana Nyfjord, Panagiotis Germanakos, and George Samaras, *Personalised Continuous Software Engineering*, Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering, 2014, pp. 57–62.

[24] Mary Poppendieck, *Lean software development: an agile toolkit*, Addison-Wesley Professional, 2003.

[25] Mary Poppendieck and Michael A Cusumano, *Lean Software Development: A Tutorial*, IEEE Software (2012), 26–32.

[26] Eric Ries, *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation To Create Radically Successful Businesses*, Crown Business, 2011.

[27] Daniel Ståhl and Jan Bosch, *Modeling Continuous Integration Practice Differences in Industry Software Development*, Journal of Systems and Software (2014), 48–59.

[28] Annika Steiber and Sverker Alänge, *A Corporate System for Continuous Innovation: The case of Google Inc.*, European Journal of Innovation Management (2013), 243–264.

[29] Diane Tang, Ashish Agarwal, Deirdre O'Brien, and Mike Meyer, *Overlapping Experiment Infrastructure: More, Better, Faster Experimentation*, Proceedings 16th Conference on Knowledge Discovery and Data Mining, 2010, pp. 17–26.

[30] Joan E. van Aken, *Management Research Based on the Paradigm of the Design Sciences: The Quest for Field-Tested and Grounded Technological Rules*, Journal of Management Studies **41** (2004), no. 2, 219–246.

[31] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén, *Experimentation in Software Engineering*, Springer, 2012.

[32] Robert Yin, *Case study research: design and methods*, 4th ed., SAGE Publications, Inc., 2009.